Language Reference 29

THINK C is a conforming implementation of the C programming language as defined in the ANSI C standard, *American National Standard for Information Systems—Programming Language—C X3.159–1989*. This chapter describes how THINK C implements those aspects of the language that the standard denotes as implementation-defined.

The section numbers in this chapter correspond to the sections in the ANSI standard.

Contents

Introduction 587
Implementation-defined behavior 587
Undefined behavior 587
Setting ANSI conformance 587
About the standard libraries 587

Language Reference 588
2.1.1.2 Translation phases 588
2.1.1.3 Diagnostics 588
2.1.2.2.1 Program startup 588
2.1.2.3 Program execution 588
2.2.1 Character sets 588
2.2.1.2 Multibyte characters 588
2.2.4.2.1 Sizes of integral types <limits.h> 588
3.1.2 Identifiers 589
3.1.2.2 Linkages of identifiers 589
3.1.2.5 Types 589
3.1.3.4 Character constants 591
3.1.7 Header names 592
3.2.1.2 Signed and unsigned integers 592
3.2.1.3 Floating and integral 592
3.2.1.4 Floating types 592
3.3 Expressions 592
3.3.2.3 Structure and union members 592
3.3.3.4 The sizeof operator 593
3.3.4 Cast operators 593
3.3.5 Multiplicative operators 593
3.3.6 Additive operators 593
3.3.7 Bitwise shift operators 593
3.3.8 Relational operators 593

3.5.1 Storage-class specifiers . . . 594
3.5.2.1 Structure and union specifiers . . . 594
3.5.2.2 Enumeration specifiers . . . 595
3.5.3 Type qualifiers . . . 595
3.5.4 Declarators . . . 595
3.6.4.2 The switch statement . . . 595
3.8.1 Conditional inclusion . . . 595
3.8.2 Source file inclusion . . . 596
3.8.3 Macro replacement. . . . 596
3.8.6 Pragma directives . . . 596
3.8.8 Predefined macro names. . . . 599
4.1.5 Common definitions <stddef.h> . . . 600
4.2 Diagnostics <assert.h> . . . 600
4.3.1 Character-testing functions . . . 600
4.5.1 Treatment of error conditions . . . 600
4.5.6.4 The fmod function . . . 600
4.7.1.1 The signal function . . . 601
4.9.2 Streams . . . 601
4.9.3 Files. . . . 601
4.9.4.1 The remove function . . . 601
4.9.4.2 The rename function . . . 601
4.9.5.2 The fflush function . . . 601
4.9.6.1 The fprintf function . . . 602
4.9.6.2 The fscanf function . . . 602
4.9.9.1 The fgetpos function . . . 602
4.9.9.4 The ftell function. . . . 602
4.9.10.4 The perror function . . . 602
4.10.3 Memory management functions . . . 602
4.10.4.1 The abort function . . . 602
4.10.4.3 The exit function . . . 602
4.10.4.4 The getenv function . . . 602
4.10.4.5 The system function . . . 603
4.11.6.2 The strerror function . . . 603
4.12.1 Components of time . . . 603
4.12.2.1 The clock function . . . 603

THINK C Extensions . . . 603
Inline assembly. . . . 603
pascal keyword. . . . 603
C++ style comments . . . 603
short double type . . . 603
Identifiers after #else and #endif . . . 604
Inline function definitions . . . 604
Low memory global definitions . . . 604
MC68881 unary inline functions . . . 604
Function prototypes . . . 604
Dimensionless arrays allowed . . . 605
void * . . . 605
Predefined symbols . . . 605

THINK C Object Extensions . . . 605
Keywords. . . . 605

Introduction

This chapter describes the way THINK C implements the behavior that the standard identifies as “implementation-defined.” It also documents consistent behavior in THINK C that the standard identifies as “undefined behavior.” These behaviors represent conforming extensions. This chapter is not a substitute for the ANSI standard.

Implementation-defined behavior

The ANSI Standard defines implementation-defined behavior as:

> “...behavior, for a correct program construct and correct data, that depends on the characteristics of the implementation and that each implementation shall document.”

Implementation-defined behavior covers such things as the way error messages are reported, the number of significant characters in identifiers, the format for integers and floating-point numbers, and so on.

Undefined behavior

The ANSI Standard defines undefined behavior as:

> “...behavior, upon use of a nonportable or erroneous program construct, of erroneous data, or of indeterminately valued objects, for which the standard imposes no requirements.”

In most cases, undefined behavior is ignored, generates a diagnosed error, or results in a run-time error. This section uses the notation “(Conforming Extension)” to mark the instances in which THINK C behaves in a predictable manner for cases that the standard specifies as undefined. If you are writing portable C code, your program should not rely on the behavior described that way.

Setting ANSI conformance

The default options settings for THINK C are not ANSI-conformant. To make THINK C an ANSI-conformant compiler, open the **THINK C options** dialog, choose the Language Settings page, and click the ANSI Settings button. Any options that affect ANSI conformance appear on that page.

29 Language Reference

About the standard libraries

For more information about implementation-defined behavior or THINK C extensions to the standard libraries described in section 4.0 of the standard, refer to the online *Standard Libraries Reference* that came with your package.

The section numbers in this chapter correspond to the sections in the ANSI standard.

Language Reference

2.1.1.2 Translation phases

The last line of a file does not need to end with a newline character. (Conforming Extension)

2.1.1.3 Diagnostics

Errors that occur during translation are reported in a Macintosh alert box or in the Compile Errors window. The offending line is highlighted in a text editing window. Link errors are reported in a Link Errors window.

When the preprocessor encounters an `#error` directive, the words "#error directive" appear in a Macintosh alert box, and the line containing the `#error` directive is highlighted in a text editing window.

2.1.2.2.1 Program startup

If the `ccommand` function is not used, `argc` is set to 1, and `argv[0]` is the empty string. If the `ccommand` function is used, `argc` and `argv` are set according to the values provided in the `ccommand` dialog box. See the description of `ccommand` in the online *Standard Libraries Reference* for more information.

2.1.2.3 Program execution

In programs that use the console package, the Console window is an interactive device. In Macintosh programs (programs that initialize the Macintosh Toolbox), the interactive device is output only.

2.2.1 Character sets

The source and execution characters include the full Macintosh character set.

2.2.1.2 Multibyte characters

There are no shift states for multibyte characters. Multibyte characters are 1 byte long.

2.2.4.2.1 Sizes of integral types <limits.h>

The number of bits in a character in the execution character set is 8.

3.1.2 Identifiers

All characters in an identifier are significant. The number of significant characters in an identifier with external linkage is 255. Case is significant in identifiers with external linkage.

3.1.2.2 Linkages of identifiers

The same identifier may have internal and external linkage. Internal linkage takes precedence over external linkage. (Conforming Extension)

```
extern int foo;      /* this foo has     */
                     /* external linkage */
static int foo = 63; /* this foo has     */
                     /* internal linkage */
```

3.1.2.5 Types

Integers are represented as two's complement binary numbers. The sizes of integer types are:

Type	Bytes
`char`	1
`short`	2
`long`	4
`int`	2 or 4

You can select the size of the type `int` in the Compiler Settings page of the **THINK C options** dialog.

The limits for the integer types are given in the header file `<limits.h>`.

THINK C uses five different representations for floating-point values. Depending on the options set, different representations map to different types. The floating-point representations are:

- 4-byte IEEE single precision
- 8-byte IEEE double precision
- 10-byte SANE extended precision
- 12-byte MC68881 extended precision
- 12-byte universal extended precision

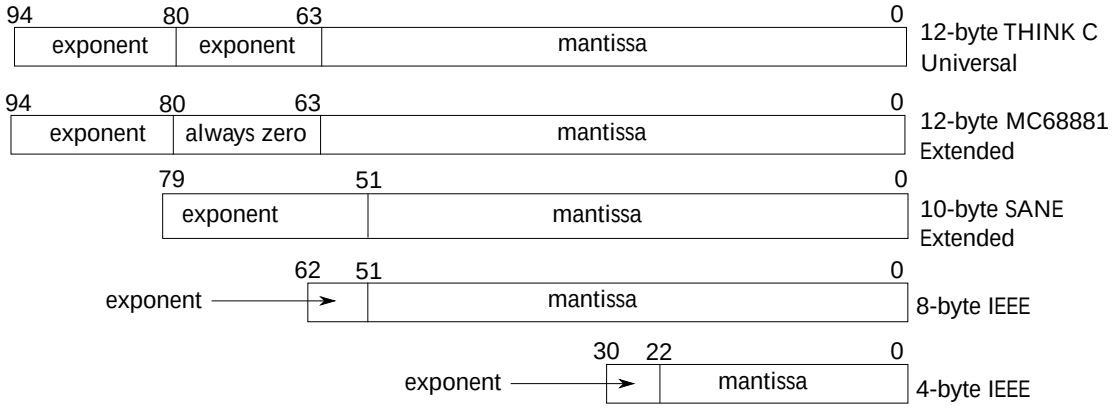

Figure 29-1 Floating-point formats

These formats, except THINK C Universal, are documented in detail in *Apple Numerics Manual, Second Edition* (Addison-Wesley) by Apple Computer and *M68000 Family Programmer's Reference Manual.*

The universal floating-point format is a THINK C extension that lets THINK C use the same format whether the program uses MC68881 code generation or SANE. The universal format is the same as the MC68881 extended format, but it duplicates the exponent part in the second word.

This table shows how floating-point types map to floating-point formats.

Type	Format	Option settings
`float`	4-byte IEEE	Any option setting
`short double`	8-byte IEEE	"THINK C" language extension on
`long double`	10-byte SANE	"Generate 68881 instructions" off, "Native floating-point format" on
`long double`	12-byte MC68881	"Generate 68881 instructions" on, "Native floating-point format" on
`long double`	12-byte universal	"Native floating-point format" off
`double`	long double	"8-byte doubles" off
`double`	8-byte IEEE	"8-byte doubles" on

Table 29-1 The floating-point format used for each type

3.1.3.4 Character constants

If a string literal begins with the sequence `\p` or `\P`, it is treated as a Pascal string. The `\p` or `\P` is replaced with the length of the string. The type of Pascal strings is `unsigned char []`, or optionally, `char []`. Pascal strings are not null-terminated.

The mapping of characters in the source character set to the execution character set is one-to-one. Multibyte characters are 1 byte long and map one-to-one with single-byte characters.

The basic execution character set consists of all 256 Macintosh characters. There are no integer character constants or escape sequences that cannot be represented in the basic execution character set.

An integer constant may contain one, two, or four characters from the execution character set. The value of a two-character constant

'AB' is 0x4142 and of type short. The value for a four-character constant 'ABCD' is 0x41424344 and of type long.

The backslash character \ is ignored for all unspecified escape sequences. (Conforming Extension)

3.1.7 Header names

In header name preprocessing, header names are treated as character strings with different delimiters. The characters ', \, ", and /* are allowed between the < and > delimiters, and the characters ', \, and /* are allowed between the " delimiters. (Conforming Extension)

3.2.1.2 Signed and unsigned integers

If an integer is converted to a shorter signed integer, the low-order bits are retained and the high-order bit of the shorter integer is treated as the sign bit.

If an unsigned integer is converted to a signed integer of equal length, and the value cannot be represented in the signed integer, there is no change in representation, but the high bit becomes the sign bit.

3.2.1.3 Floating and integral

When a value of an integral type is converted to a floating type, if the value being converted is in the range of values that cannot be represented exactly, the result is rounded in the current rounding mode. The default rounding mode is round to nearest, and may be changed by the program.

3.2.1.4 Floating types

If a floating value being converted is in the range of values that can be represented, but cannot be represented exactly, the result is rounded in the current rounding mode. The default rounding mode is round to nearest, and may be changed by the program.

3.3 Expressions

Bitwise operations on signed integers are carried out as if they were unsigned.

3.3.2.3 Structure and union members

If a member of a union object is accessed using a member of a different type, the data stored at that location is treated as if it were a member of the accessing type.

3.3.3.4 The sizeof operator

The type of the `sizeof` operator is `size_t`, which is defined as `unsigned long`.

3.3.4 Cast operators

A pointer to a function may be converted to a pointer to an object without losing the value of the pointer. (Conforming Extension)

When converting from a pointer to an integer, if the integer is 4 bytes, then all the bits are used without a change of representation; the 4-byte quantity is interpreted as an integer whose type (for example, signed or unsigned) is given by the cast expression. If the integer is smaller than a pointer, then the low-order bytes are used.

3.3.5 Multiplicative operators

The sign of the remainder is the same as the sign of the dividend.

If a and b are…		Then a/b and a%b are…	
a=	b=	a/b=	a%b=
5	3	1	2
-5	3	-1	-2
5	-3	-1	2
-5	-3	1	-2

Table 29-2 Results of division and modulus operators

3.3.6 Additive operators

Adding to or subtracting from a pointer that does not behave like a pointer to an element of an array is allowed. Memory is treated as a linear address space. Pointers that do not behave as if they point to the same array object may be subtracted. (Conforming Extension)

The type of integer required to hold the difference between two pointers is `ptrdiff_t`, which is defined as `long`.

3.3.7 Bitwise shift operators

A right shift of a negative-valued signed integral type copies the sign bit in.

3.3.8 Relational operators

Pointers may be compared using a relational operator even if they do not point to the same aggregate or union. (Conforming Extension)

3.5.1 Storage-class specifiers

The way THINK C allocates registers depends on the settings in the Code Optimization page of the **THINK C options** dialog.

If the Automatic Register Assignment option is off, the following registers are available to register allocation:

Registers	Used for
A2-A4	Pointer types
D3-D7	Integral types, pointer types, and floating-point types if the Generate 68881 instructions option is on
FP4-FP7	12-byte floating-point types if the Generate 68881 instructions option is on

Table 29-3 Registers available for allocation

If the project is not an application (code resource, device driver, or desk accessory), register A4 is not available for register allocation.

If the Automatic Register Assignment option is on, and the Honor 'register' first suboption is on, THINK C places as many variables declared register into registers as described above.

The order that registers are allocated in is unspecified. Structs and unions are never placed in a register, even if they would fit into one.

3.5.2.1 Structure and union specifiers

A bit-field may be declared with any integral type. The size of the declared type determines the "word" size for that bit-field, so a "word" may be 8, 16, or 32 bits wide.

A sequence of bit-fields with the same word size are packed into a word. No bit-field may be wider than its word size. If a bit-field would straddle a word boundary, it is placed in the next word.

Bit-fields are assigned beginning with the high-order bit of a word. An unnamed field with a width of 0 "closes out" the current word. A bit-field with a different word size from the preceding bit-field causes this closing out to happen automatically, just as a non-bit-field member does. (Conforming Extension)

A plain `int` bit-field is treated as a `signed int`.

All structures and unions are padded if necessary to be even-sized. Padding is inserted after a member only to meet the alignment requirements of the next member or at the end of the structure or union. Only odd-sized data items (for example, `char`, odd-sized array of `char`, `char`-size `enum` types, and so on) do not need to be aligned.

3.5.2.2 Enumeration specifiers

Enumerated types are of type `int` if the Enums Are Always Ints option is on. Otherwise, the type of an enumerated type is the first one in the following list that is sufficient to hold all the values:

"4-byte ints" off	"4-byte ints" on
`char`	`char`
`int`	`short`
`long`	`int`

Table 29-4 The type used for an enumerated type

3.5.3 Type qualifiers

In a compound assignment to a volatile-qualified type, THINK C may generate a fetch and a store. Otherwise, each use of the volatile-qualified type results in a fetch or a store. If the volatile object is larger than 4 bytes, the compiler may need to generate multiple fetches or multiple stores.

3.5.4 Declarators

The maximum number of declarators is at least 12 but usually significantly greater. The actual number depends on the particular combination of declarators.

3.6.4.2 The switch statement

The maximum number of case values in `switch` statements is limited only by available memory.

3.8.1 Conditional inclusion

The token `defined` has special meaning only if it appears literally with an `#if` or `#elif` directive. (Conforming Extension)

The value of a single-character constant in a constant expression that controls conditional inclusion matches the value of the same character constant in the execution character set. Such character constants can have a negative value.

3.8.2 Source file inclusion

The mechanism that THINK C uses to locate header files is described in "How THINK C looks for header files" in Chapter 15.

The name in the `#include` directive is interpreted as a Macintosh file name. Case is not significant, and colons are treated as volume and directory separators.

3.8.3 Macro replacement

A macro argument that consists of no preprocessing tokens is treated as no tokens. (Conforming Extension)

3.8.6 Pragma directives

THINK C implements the following `#pragma` directives:

```
once                    parameter
options                 nooptimize
```

pragma once

When this directive appears in a header file, THINK C includes it only once, even if there is more than one `#include` directive that includes the header file.

```
#pragma once
```

pragma parameter

This directive applies to a subsequent inline function definition and allows parameters to be passed in registers instead of on the stack. It specifies which register holds the return value and which registers parameters are passed in. The `#pragma parameter` directive must appear before the inline declaration.

`#pragma parameter` *return-regopt function-name (param-reglistopt)*

Note

Inline function definitions are not normally part of the ANSI-conformant implementation of THINK C, but are enabled as a conforming extension if a `#pragma parameter` directive appears. For more information about inline function definitions, see "Inline function definitions" in Chapter 15.

Function-name is the name of a function that is subsequently defined inline. If the definition is not an inline definition, never defined, or already defined, the directive is ignored.

The optional *return-reg* can be `__A0`, `__A1`, `__D0`, or `__D1`.

The optional *param-reglist* is a parameter list made up of `__A0`, `__A1`, `__D0`, `__D1`, or `__D2`.

The inline definition must have a prototype, and the prototype must not end with `...`. The return type must be an integer type or a pointer type. The return type must be 4 bytes long if the return is in an address register. The argument types may not exceed 4 bytes, except when the inline definition is declared `pascal`, in which case the address of the parameter is used. An address register can hold a 2-byte value for arguments only. In no case can an address register hold a 1-byte value.

Examples:

```
#pragma parameter __A0 NewHandleClear(__D0)
pascal Handle NewHandleClear( Size
    byteCount )
 = 0xA322;

#pragma parameter Delay(__A0,__A1)
pascal void Delay( long numTicks,
   long *finalTicks )
 = {0xA03B,0x2280};
```

pragma options
This directive allows programmers to turn certain options on and off from within your source code.

```
#pragma options (option-list)
```

Option-list is a comma-separated list of one of the following identifiers. If the identifier is given, the option is turned on. If !*identifier* is given, the option is turned off.

Option	Meaning if on
`signed_pstrs`	`"\p..."` is `char []`
`align_arrays`	Align even-length `char` arrays within structures
`virtual`	Methods are virtual by default
`indirect`	Classes are indirect by default
`pack_enums`	Select appropriate size for `enum` types
`trigraphs`	Decode trigraphs
`require_protos`	Function prototypes are required
`infer_protos`	Infer function prototypes
`check_ptrs`	Check pointer types
`mc68020`	Generate code for 68020
`mc68881`	Generate code for 68881
`macsbug_names`	Generate MacsBug names
`long_macsbug_names`	Use long format for MacsBug names
`profile`	Generate profiling calls
`honor_register`	Honor register declarations
`force_frame`	Always generate stack frames
`assign_registers`	Assign variables to registers
`defer_adjust`	Defer and combine stack adjusts
`redundant_loads`	Suppress redundant loads

Table 29-5 Identifiers for the options pragma directive

Option	Meaning if on
`global_optimizer`	Run global optimizer
`gopt_induction`	(global optimizer) Induction variables
`gopt_cse`	(global optimizer) Common subexpression elimination
`gopt_loop`	(global optimizer) Loop invariants
`gopt_coloring`	(global optimizer) Register coloring

Table 29-5 Identifiers for the options pragma directive *(Continued)*

For more information about using the `options` pragma directive, see "Accessing Option Settings in Your Code" in Chapter 15.

pragma nooptimize
The `nooptimize` pragma directive allows the appearance of a call to a function to disable the global optimizer for the function that makes the call.

```
#pragma nooptimize(function-name)
```

Function-name must have already been declared as a function. Otherwise, it is an error.

Any function that calls a function that has been the subject of the `nooptimize` directive is not optimized by the global optimizer, even if the option is on. For an example, look at the implementation of `setjmp`.

This directive should only be used for functions that have atypical flow-of-control properties.

3.8.8 Predefined macro names

The following predefined macro names can be the subject of `#undef` and `#define` preprocessing directives: `__LINE__`, `__FILE__`, `__DATE__`, `__TIME__`, and `__STDC__`. They must be undefined with `#undef` before they can be defined with `#define`. (Conforming Extension)

The identifier `defined` is interpreted specially in context in `#if` or `#elif` preprocessing directives. It can be the subject of `#define` and `#undef` preprocessing directives. (Conforming Extension)

__DATE__ and __TIME__ are always available.

4.1.5 Common definitions <stddef.h>

The following types are defined in <stddef.h>

Type	Definition
NULL	((void *) 0)
size_t	unsigned long
ptrdiff_t	long

Table 29-6 Common types defined in stddef.h

4.2 Diagnostics <assert.h>

The diagnostic printed by assert is of the form "Assertion failed: *expression*, file *xyz*, line *nnn*". The message appears on stderr, and, if possible, is followed by the message "press return to exit".

4.3.1 Character-testing functions

The character-testing functions operate on the ranges in Table 29-7.

Function	Range
isalnum()	a-z, A-Z, 0-9
isalpha()	a-z, A-Z
iscntrl()	0x00-0x1F, 0x7F
islower()	a-z
isprint()	0x20-0x7E
isupper()	A-Z

Table 29-7 Character-testing functions

4.5.1 Treatment of error conditions

The values returned by mathematics functions on domain errors are documented for each function in the online Standard Libraries Reference.

The mathematics functions do not set errno to ERANGE on underflow range errors.

4.5.6.4 The fmod function

If the second argument to fmod is 0, fmod returns 0 and sets errno to EDOM.

4.7.1.1 The signal function

The set of signals, their semantics, and the default handling of signals are described in the documentation of `signal` in the online Standard Libraries Reference.

No implementation-defined blocking of signals is performed. The default handler is reset if a `SIGILL` signal is received.

4.9.2 Streams

The last line of a text stream does not require a terminating newline character. Trailing blanks are not stripped off when writing to a text stream. No null characters are appended to a binary stream.

4.9.3 Files

The file position indicator of an append mode stream initially points to the end of the file. Zero-length files actually exist. A write on a text stream does not truncate beyond that point. Writing in place is allowed.

THINK C implements buffered and unbuffered I/O. Line-buffered I/O is the same as fully buffered I/O.

The rules for composing valid file names are identical to Macintosh file-naming rules. In general, file names may have up to 31 characters, and the `:` character separates directory names. See the online Standard Libraries Reference for more information.

The same file can be opened more than once if the Macintosh file system allows it.

4.9.4.1 The remove function

It is an error to remove an open file.

4.9.4.2 The rename function

It is an error to use `rename` to give a file a name that already exists.

4.9.5.2 The fflush function

If you use `fflush` on a stream that is being used for input or update, and the most recent operation was input, the buffer is cleared and the next read will go to disk. (Conforming Extension)

The `fflush` function always returns the buffer to a neutral state, so this function can be used to switch between input and output, regardless of the last I/O operation. (Conforming Extension)

4.9.6.1 The fprintf function

If the conversion specification for the `fprintf` function contains a # flag and the conversion specifier is `s`, the matching argument is treated as a Pascal string. (Conforming Extension)

The output for the `%p` conversion in `fprintf` is the same as `%.8lX`, eight digit, hexadecimal integer that uses capital letters and leading zeros.

4.9.6.2 The fscanf function

The input format for the `%p` conversion in `fscanf` is the same as `%lx`, a hexadecimal integer.

A dash (`-`) character that is neither the first nor the last character in the scanlist for `%[` conversion is treated as a range specifier. For example, `%[0-9]` means the range of characters from `'0'` to `'9'` inclusive.

4.9.9.1 The fgetpos function

If `fgetpos` fails, `errno` is set to `ENODEV`.

4.9.9.4 The ftell function

If `ftell` fails, `errno` is set to `ENODEV.`

4.9.10.4 The perror function

The function `perror` prints the supplied error string, the string `"Error: "`, and the value of `errno`.

4.10.3 Memory management functions

If the number of bytes of memory requested from `calloc`, `malloc`, or `realloc` is 0, these functions return a pointer to a unique zero-size block of memory.

4.10.4.1 The abort function

The `abort` function closes all open and temporary files.

4.10.4.3 The exit function

The argument to the `exit` function is always ignored.

4.10.4.4 The getenv function

The `getenv` function always returns `NULL.`

4.10.4.5 The system function

The `system` function always ignores its argument and always returns 0 (zero).

4.11.6.2 The strerror function

The string that `strerror` returns is `"Error: "` and the value of its argument.

4.12.1 Components of time

The time functions are not aware of daylight savings time or the local time zone.

4.12.2.1 The clock function

The `clock` function returns the number of ticks (60ths of a second) since the Macintosh was turned on.

THINK C Extensions

This section describes the extensions to ANSI C that are enabled when you set the Language Extensions option in the **THINK C options** dialog.

Inline assembly

The identifier `asm` is reserved for the inline assembler. For more information about the inline assembler, see Chapter 20, “Writing Inline Assembly.”

pascal keyword

The identifier `pascal` is reserved to define functions that follow Pascal calling conventions. For examples, see “Calling Toolbox Routines” in Chapter 18. For a description of Pascal calling conventions, see “Calling Pascal Routines” in Chapter 19.

C++ style comments

Two slashes (`//`) introduce a comment. The comment ends at the newline character.

short double type

The type `short double` is allowed. It is an 8-byte IEEE floating-point number.

Identifiers after #else and #endif

An identifier after an `#endif` or `#else` preprocessing directive is ignored.

```
#ifdef DEBUG
   printf( "oops!\n");
#endif DEBUG
```

Inline function definitions

Inline function definitions are allowed. The `#pragma parameter` directive may be used to assign registers to parameters. An inline function definition consists of either one hexadecimal literal or a series of comma-separated hexadecimal literals enclosed in braces. The hexadecimal literals represent machine language instructions.

```
#pragma parameter __A0 NewHandleClear(__D0)
pascal Handle NewHandleClear( Size
    byteCount )
 = 0xA322;

pascal void PrOpen(void)
   = {0x2F3C,0xC800,0x0000,0xA8FD};
```

When the compiler generates code for an inline function, it merely inserts the machine code.

Low memory global definitions

Global variables may be defined to refer to absolute memory addresses.

```
extern short MemErr : 0x220;
```

This construct is only legal in global scope. The `extern` storage-class specifier is optional. It was required in earlier versions of THINK C.

MC68881 unary inline functions

Unary inline functions for the MC68881 may be defined like this:

```
double atanh (double) : 0x0D;
```

The value after the colon is used as the 7 bits of the second word of the MC68881 instruction. See `<math.h>` for examples.

Function prototypes

The parameter list `(...)` is allowed in a function prototype.

Dimensionless arrays allowed

Dimensionless arrays are allowed as the last member of `struct` definitions. The member does not contribute to the size of the `struct`. For example:

```
struct {
   short count;
   char data[];
} CountData;   /* sizeof(CountData) is 2 */
```

void *

The type `void *` is compatible with function pointers and data pointers.

Predefined symbols

The preprocessor symbol `THINK_C` is defined as `6` in THINK C 6.0 and 6.5, and as `5` in THINK C 5.0. It is defined as `1` in THINK C 4.0. It is not defined at all for prior versions.

THINK C Object Extensions

This section summarizes the extensions to ANSI C when the Language Extensions option is set to THINK C + Objects. For more information about the way THINK C implements objects, see Chapter 23, “Using Objects in THINK C.”

Keywords

When the THINK C + Objects language extension option is on, THINK C reserves the following names as keywords:

```
class                delete
new                  virtual
```

The following names are interpreted specially in context, but they are not keywords:

```
direct        indirect      inherited
operator      private       protected
public        this
```